6570P019/2003P00348 *Patent*

UNITED STATES PATENT APPLICATION

for

STARTUP FRAMEWORK AND METHOD FOR ENTERPRISE COMPUTING SYSTEMS

INVENTORS:

Ingo Zenz
Krasimir Semerdzhiev

Prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN, LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CALIFORNIA 90025
(408) 720-8598

Attorney's Docket No. *6570P019/2003P00348*

A STARTUP FRAMEWORK AND METHOD FOR ENTERPRISE COMPUTING SYSTEMS

BACKGROUND

Field of the Invention

[0001] This invention relates generally to the field of data processing systems. More particularly, the invention relates to an improved startup framework and method for an enterprise computing system.

Description of the Related Art

[0002] Traditional client-server systems employ a two-tiered architecture such as that illustrated in **Figure 1a**. Applications 102 executed on the client side 100 of the two-tiered architecture are comprised of a monolithic set of program code including a graphical user interface component, presentation logic, business logic and a network interface that enables the client 100 to communicate over a network 103 with one or more servers 101. A database 104 maintained on the server 101 provides non-volatile storage for the data accessed and/or processed by the application 102.

[0003] As is known in the art, the "business logic" component of the application represents the core of the application, i.e., the rules governing the underlying business process (or other functionality) provided by the application. The "presentation logic" describes the specific manner in which the results of the business logic are formatted for display on the user interface. The "database"

104 includes data access logic used by the business logic to store and retrieve data.

[0004] The limitations of the two-tiered architecture illustrated in **Figure 1a** become apparent when employed within a large enterprise. For example, installing and maintaining up-to-date client-side applications on a large number of different clients is a difficult task, even with the aid of automated administration tools. Moreover, a tight coupling of business logic, presentation logic and the user interface logic makes the client-side code very brittle. Changing the client-side user interface of such applications is extremely hard without breaking the business logic, and vice versa. This problem is aggravated by the fact that, in a dynamic enterprise environment, the business logic may be changed frequently in response to changing business rules. Accordingly, the two-tiered architecture is an inefficient solution for enterprise systems.

[0005] In response to limitations associated with the two-tiered client-server architecture, a multi-tiered architecture has been developed, as illustrated in **Figure 1b**. In the multi-tiered system, the presentation logic 121, business logic 122 and database 123 are logically separated from the user interface 120 of the application. These layers are moved off of the client 125 to one or more dedicated servers on the network 103. For example, the presentation logic 121, the business logic 122, and the database 123 may each be maintained on separate servers, 126, 127 and 128, respectively. In fact, depending on the size

of the enterprise, each individual logical layer may be spread across multiple dedicated servers.

[0006] This division of logical components provides a more flexible and scalable architecture compared to that provided by the two-tier model. For example, the separation ensures that all clients 125 share a single implementation of business logic 122. If business rules change, changing the current implementation of business logic 122 to a new version may not require updating any client-side program code. In addition, presentation logic 121 may be provided which generates code for a variety of different user interfaces 120, which may be standard browsers such as Internet Explorer® or Netscape Navigator®.

[0007] The multi-tiered architecture illustrated in **Figure 1b** may be implemented using a variety of different application technologies at each of the layers of the multi-tier architecture, including those based on the Java 2 Enterprise Edition™ ("J2EE") standard, the Microsoft .NET standard and/or the Advanced Business Application Programming ("ABAP") standard developed by SAP AG. For example, in a J2EE environment, the business layer 122, which handles the core business logic of the application, is comprised of Enterprise Java Bean ("EJB") components with support for EJB containers. Within a J2EE environment, the presentation layer 121 is responsible for generating servlets and Java Server Pages ("JSP") interpretable by different types of browsers at the user interface layer 120.

[0008] Although the multi-tiered system illustrated in **Figure 1b** provides a more flexible and scalable architecture, it also results in significant additional complexity. For example, managing multiple clusters of presentation layer servers, business layer servers and databases, and the dependencies between them requires a significant amount of network administration overhead. Accordingly, new techniques which simplify the startup and management of multi-tiered enterprise systems are desirable.

SUMMARY

[0009] A system and method are described for managing a plurality of sever nodes. In one embodiment, the sever nodes are organized into groups referred to as "instances." Each instance includes a group of redundant sever nodes and a dispatcher for distributing service requests to the sever nodes. In one embodiment, a hierarchical configuration data object is stored within a database and is centrally accessible by all of the servers from all of the instances. The hierarchical configuration data object organizes configuration data and binary data in a manner which simplifies sever node management in a large enterprise network. For example, in one embodiment of the invention, when starting up servers and dispatchers within an instance, the server layout of the instance is retrieved from the hierarchical configuration data object. In addition, binaries and configuration parameters stored locally on the servers/dispatcher are initially compared with the binaries and configuration parameters stored within the hierarchical configuration data object. If the binaries and configuration parameters stored locally are out-of-date, then new binaries/parameters are downloaded from a central database.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] A better understanding of the present invention can be obtained from the following detailed description in conjunction with the following drawings, in which:

[0011] **FIG. 1a** illustrates a traditional two-tier client-server architecture.

[0012] **FIG. 1b** illustrates a prior art multi-tier client-server architecture.

[0013] **FIG. 2** illustrates a sever node system architecture according to one embodiment of the invention.

[0014] **FIG. 3** illustrates a sever node system architecture which employs a configuration data caching.

[0015] **FIG. 4** illustrates a method for synchronizing configuration data according to one embodiment of the invention.

[0016] **FIG. 5** illustrates a hierarchical representation of configuration data employed in one embodiment of the invention.

[0017] **FIG. 6** illustrates a plurality of property sheet data object employed within a configuration hierarchy according to one embodiment of the invention.

[0018] **FIGS. 7a-c** illustrate one embodiment of a property sheet which retains a default value for each parameter even when a custom value is entered.

[0019] **FIG. 8** illustrates a method for starting up one or more instances according to one embodiment of the invention.

[0020] **FIG. 9** illustrates one embodiment of a startup framework for implementing the method illustrated in **FIG. 8**.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

[0021] Described below is a system and method for managing multiple sever node clusters using a central database arrangement. Throughout the description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without some of these specific details. In other instances, well-known structures and devices are shown in block diagram form to avoid obscuring the underlying principles of the present invention.

EMBODIMENTS OF THE INVENTION

[0022] A system architecture according to one embodiment of the invention is illustrated in **Figure 2**. The architecture includes a central services instance 200 and a plurality of application server instances 210, 220. As used herein, the application server instances, 210 and 220, each include a group of sever nodes 214, 216, 218 and 224, 226, 228, respectively, and a dispatcher, 212, 222, respectively. The central services instance 200 includes a locking service 202 and a messaging service 204 (described below). The combination of all of the application instances 210, 220 and the central services instance 200 is referred to herein as a "cluster." Although the following description will focus solely on instance 210 for the purpose of explanation, the same principles apply to other instances such as instance 220.

[0023] The sever nodes 214, 216, 218 within instance 210 provide the business and/or presentation logic for the network applications supported by the system. Each of the sever nodes 214, 216, 218 within a particular instance 210 may be configured with a redundant set of application logic and associated data. In one embodiment, the dispatcher 210 distributes service requests from clients to one or more of the sever nodes 214, 216, 218 based on the load on each of the servers. For example, in one embodiment, the dispatcher 210 implements a round-robin policy of distributing service requests.

[0024] The sever nodes 214, 216, 218 may be Java 2 Enterprise Edition ("J2EE") sever nodes which support Enterprise Java Bean ("EJB") components and EJB containers (at the business layer) and Servlets and Java Server Pages ("JSP") (at the presentation layer). Of course, the embodiments of the invention described herein may be implemented in the context of various different software platforms including, by way of example, Microsoft .NET platforms and/or the Advanced Business Application Programming ("ABAP") platforms developed by SAP AG, the assignee of the present application.

[0025] In one embodiment, communication and synchronization between each of the instances 210, 220 is enabled via the central services instance 200. As illustrated in **Figure 2**, the central services instance 200 includes a messaging service 204 and a locking service 202. The message service 204 allows each of the servers within each of the instances to communicate with one another via a message passing protocol. For example, messages from one

server may be broadcast to all other servers within the cluster via the messaging service 204 (e.g., such as the cache configuration messages described below). Alternatively, messages may be addressed directly to specific servers within the cluster (i.e., rather than being broadcast to all servers).

[0026] In one embodiment, the locking service 202 disables access to (i.e., locks) certain specified portions of configuration data and/or program code stored within a central database 230. The locking manager locks data on behalf of various system components which need to synchronize access to specific types of data and program code (e.g., such as the configuration managers 244, 254 illustrated in **Figure 2**). As described in detail below, the locking service enables a distributed caching architecture for caching copies of server/dispatcher configuration data.

[0027] In one embodiment, the messaging service 204 and the locking service 202 are each implemented on dedicated servers. However, the messaging service 204 and the locking service 202 may be implemented on a single server or across multiple servers while still complying with the underlying principles of the invention.

[0028] As illustrated in **Figure 2**, each sever node (e.g., 218, 228) includes a lock manager 240, 250 for communicating with the locking service 202; a cluster manager 242, 252 for communicating with the messaging service 204; and a configuration manager 244, 254 for communicating with a central database 230

(e.g., to store/retrieve configuration data as described herein). Although the lock manager 240, 250, cluster manager 242, 252 and configuration manager 244, 254 are illustrated only with respect to sever nodes 218 and 228 in **Figure 2**, each of the sever nodes 214, 216, 224 and 226 and/or on the dispatchers 212, 222 may be equipped with equivalent lock managers, cluster managers and configuration managers while still complying with the underlying principles of the invention.

One Embodiment of a Hierarchical Configuration Data Object

[0029] Referring now to **Figure 3**, in one embodiment, configuration data 320 defining the configuration of the central services instance 200 and/or the sever nodes and dispatchers within instances 210 and 220, is stored within the central database 230. By way of example, the configuration data may include an indication of the kernel, applications and libraries required by each dispatcher and server; network information related to each dispatcher and server (e.g., address/port number); an indication of the binaries required during the boot process for each dispatcher and server, parameters defining the software and/or hardware configuration of each dispatcher and server (e.g., defining cache size, memory allocation, . . . etc), and various other types of information related to the cluster. It should be noted, however, that the underlying principles of the invention are not limited to any particular set of configuration data.

[0030] In one embodiment of the invention, to improve the speed at which the various servers and dispatchers access the configuration data, the configuration

managers 244, 254 cache configuration data locally within configuration caches 300, 301. As such, to ensure that the configuration data within the configuration caches 300, 301 remains up-to-date, the configuration managers 244, 254 implement cache synchronization policies, as described herein.

[0031] One embodiment of a method for synchronizing configuration data across each of the application server instances 210, 220 and the central services instance 200 is illustrated in **Figure 4**. It is assumed initially that certain portions of the configuration data from the central database is cached locally within configuration cache 300 and configuration cache 301.

[0032] At 400 (**Figure 4**), a user or network administrator attempts to modify the configuration data stored within configuration cache 300 on sever node 228. In response, the configuration manager 254 attempts to acquire a lock on the relevant portions of the configuration data by transmitting a lock request to the locking service 202. If the configuration data is not being currently modified by another transaction, then the locking service locks the configuration data on behalf of the configuration manager 254. Once locked, the configuration managers 244 of other sever nodes 218 will not be permitted to access or acquire a lock on the configuration data.

[0033] At 402, the configuration data within the configuration cache 300 of sever node 228 is modified. At 404, the cluster manager 252 broadcasts an indication of the modified data to the cluster manager 242 on sever node 218

and the cluster manager of other sever nodes (i.e., via the messaging service 204). At 406, the modifications to the configuration data are committed to the central database 230. In one embodiment, the modifications to the configuration data are registered within a hierarchical configuration data object 500, described below with respect to **Figure 5**.

[0034] At 408, the cluster manager 252 notifies the cluster manager 242 on sever node 218 and the cluster managers of other sever nodes of the central database update. In response, the configuration manager 244 invalidates the modified configuration data from its cache 301 and, at 412, loads the new configuration data from the central database 230. In one embodiment, the configuration manager 244 only downloads the portion of the configuration data which has been modified (i.e., rather than the entire set of configuration data). To determine whether it needs to update its configuration data, the configuration manager 244 compares the version of its configuration data with the version of the configuration data stored the central database. For example, in one embodiment, it compares an index of the data stored locally on the server/dispatcher and the hierarchical configuration data object illustrated in **Figure 5** to determine whether the configuration data at the server/dispatcher is out-of-date.

[0035] The method set forth in **Figure 4** may be rearranged in a variety of ways while still complying with the underlying principles of the invention. For example, in one embodiment, the configuration manager 244 may delete the

configuration data within its cache 301 as soon as the indication of the modification is transmitted at 404. Similarly, the locking service 202 may release the lock on the configuration data as soon as the changes to the configuration data are committed to the central database at 406. Various other reorganizations of the method elements from **Figure 4** are also contemplated within the scope of the present invention.

[0036] As illustrated in **Figure 5**, in one embodiment, a hierarchical data object 500 stored within the central database 230 is used to manage the various different types of configuration data employed within the cluster. At the root of the hierarchy is the "cluster data" object 501. In one embodiment, the cluster data object 501 includes two primary sub-configuration objects: a "global settings" object 510 and an "instance settings" object 520. As suggested by its name, the global settings object 510 contains, within its logical hierarchy, global configuration data associated with all servers in the cluster (e.g., all sever node, dispatchers, and servers within the central services instance 200). Conversely, the instance settings object 520 contains server-specific configuration data (i.e., specific to each individual instances, dispatchers and servers).

[0037] In one embodiment, multiple sub-configuration objects are provided beneath the global settings object 510, for example, a "bootstrap" object 512; a "dispatchers" object 515 and a "servers" object 518. The bootstrap object 512 contains data associated with the cluster startup or "boot" process. Specifically, in one embodiment, the bootstrap object 512 contains a "binaries" object 513,

which contains an indication of the binaries (i.e., program code) required during the boot process, and a "configuration" object 514 containing parameters associated with the boot process. By way of example, and not limitation, in a Java implementation, the binaries indicated within the binaries object 513 may identify the program code required for starting the Java virtual machine and the configuration object 514 may include the parameters to be used when initializing the virtual machine.

[0038] The "dispatchers" object 515 includes a "binaries" object 516 identifying the common binaries required by each of the dispatchers within the cluster, and a "configuration" object 517 identifying the common parameters associated with all of the dispatchers in the cluster. Similarly, the "servers" object 518 includes a "binaries" configuration object 519 identifying the common binaries required by each of the sever nodes within the cluster, and a configuration object 511 identifying the common parameters associated with each of the sever nodes within the cluster.

[0039] As mentioned above, the "instance settings" object 520 contains server/instance-specific configuration settings. In one embodiment, a plurality of "instance" sub-configuration objects, such as sub-configuration objects 522, 532, and 542, are listed under the instance settings object 520, each of which identify a particular instance within the cluster. By way of example, the "Instance 01" object 522 illustrated in **Figure 5** includes an "instance globals" object 524 and an "individual servers" object 527. The instance globals object 524 identifies

binaries and settings which apply to all dispatchers 525 and/or servers 526 within the specified instance ("Instance 01"). By contrast, the "individual servers" object 527 identifies the server layout of each instance and contains server-specific settings for each dispatcher 528 and server 529, 530 within the specified instance. For example, the "individual servers" object 527 defines the number of servers within the instance and identifies each server and dispatcher with a unique network ID and network port (i.e., for communicating with the server/dispatcher).

[0040] Moreover, in one embodiment, the various "configuration" objects and "binaries" objects described herein may be further sub-divided into sub-configuration objects defining specific types of configuration parameters and binaries. For example, the "binaries" objects may include sub-configuration objects for the OS libraries (e.g., Java classfile libraries) and application-specific libraries required on each server. Similarly, the "configuration" objects may include sub-configuration objects which logically subdivide the configuration parameters into logical categories. By way of example, and not limitation, **Figure 6** illustrates one embodiment in which a "configuration" object 601 includes sub-configuration objects for interfaces 602, kernel 603 and services 604. In a Java environment, for example, the kernel of the J2EE Engine forms the bases for all services. It implements resource management (e.g., Thread management, Class loading Strategy, Configuration Management Framework, . . . etc) on top of the Java Virtual Machine. Part of the kernel configuration may also include configuration of the underlying Java VM. In one embodiment,

services 604 include those services defined by the Java 2 Enterprise Edition (J2EE) standard as well as additional proprietary services, and the interfaces 602 allow various J2EE components to communicate with one another.

One Embodiment of a Property Sheet

[0041] The kernel object 603 and services object 604 illustrated in **Figure 6** each include a plurality of "property sheet" objects 610 and 620, respectively, positioned at the bottom of the configuration hierarchy. The property sheet objects 610, 620 contain the underlying configuration parameters/arguments used for each server/dispatcher within the cluster. Specifically, in one embodiment, each property sheet comprises a plurality of name-value pairs representing a name and respective value (or set of values) for each parameter.

[0042] In one embodiment of the invention, each parameter name within the property sheet is associated with a default value and (potentially) a custom value. **Figure 7a** illustrates an exemplary view of a property sheet 700 which includes a name column 701 containing a list of parameter names, a value column 702 containing the current value associated with each of the parameter names and a column containing "custom" check-boxes 703 which indicate whether the current value of each parameter is the default value or a custom value.

[0043] In one embodiment, in response to a user selecting one of the check-boxes 703, a data entry window 710 such as that illustrated in **Figure 7b** is

generated. The window 710 includes a field for the parameter name 711, a field for the data type of the parameter 712 (e.g., integer, string, . . . etc), a field containing the default value of the parameter 713, and a "custom" field 714 into which a custom parameter value may be entered by an end user.

[0044] As illustrated in **Figure 7c**, if a custom value is entered in the custom value field 714, a check mark 720 appears in the row associated with the parameter value. Even when a custom value is entered, however, in one embodiment of the invention, the default value associated with the parameter is always preserved. As such, if an end user modifies a particular value within a particular configuration object, the default value can always be recovered (e.g., for troubleshooting purposes).

[0045] Using a central, hierarchical configuration object for managing configuration data as described above is beneficial in a variety of circumstances. For example, the configuration manager 244, 254 on each server may determine whether to invalidate a particular set of configuration data which has been modified within the cache of another server based on the location of the configuration data within the hierarchy. If the configuration data is located within the "global settings" object, for example, then the configuration manager may invalidate the data (i.e., because it applies to every server within the cluster). By contrast, if the configuration data is located within the "individual servers" object, then the configuration manager may skip invalidation of the data (i.e., because it only applies to a specific server within a specific instance). Of course, the

hierarchical representation may be used for a variety of different network management applications while still complying with the underlying principles of the invention.

One Embodiment of a System and Method for Starting Instances

[0046] One embodiment of the invention employs a unique startup framework for starting and stopping the various server instances within the cluster. **Figures 8** and **9** illustrate an exemplary startup method and framework, respectively. The startup framework 900 includes startup and control logic 902 and bootstrap logic 901. In one embodiment, the startup and control logic 902 provides the central point of control for the instance 210 and for all processes 903 executed within the servers and dispatchers of the instance 210. For example, the instance startup procedure described herein may be performed under the control of the startup and control logic 902.

[0047] Turning to the method illustrated in **Figure 8**, at 800, the startup and control logic 902 launches the bootstrap logic 901 to initiate startup of the instance (e.g., in response to a startup command entered by a network administrator). As illustrated in **Figure 9**, the bootstrap logic 901 is comprised of bootstrap binaries 513 stored within the central database 230 and the configuration of the bootstrap logic 901 is defined by configuration parameters 512 stored within the central database 230 (i.e., within the configuration data hierarchy 320). Thus, if necessary, the bootstrap logic 901 may be

modified/updated at the central database 230 and subsequently distributed to servers/instances during the startup process.

[0048] At 802, the bootstrap logic 901 retrieves up-to-date configuration data 320 from the central database 230 including the layout of the instance 210 (e.g., identifying the servers and dispatchers to be started) and the parameters/ arguments to be used for each server and dispatcher within the instance 210. In one embodiment, the bootstrap logic 901 uses this information to construct a description of the instance, which it provides to the startup and control logic 902 in the form of an "Instance Properties" data object. In one embodiment, the Instance Properties data object is a text file containing a list of servers/ dispatchers and associated parameters which the startup and control logic 902 parses to determine the instance layout and instance settings. However, various alternate data formats may be employed for the Instance Properties file while still complying with the underlying principles of the invention (e.g., such as the "Extensible Markup Language" or "XML" format).

[0049] At 804, using the instance layout and parameters/arguments from the Instance Properties data object, the startup and control logic 902 builds a list of servers and/or dispatchers to be started, including an indication of the specific instance processes to be started on each server/dispatcher. In one embodiment of the invention, at 806 (**Figure 8**) prior to starting each server/dispatcher identified in the list, the startup and control logic 902 launches node-specific bootstrap logic 905 to synchronize the binaries and configuration settings on

each server and/or dispatcher. For example, depending on how long the instance 210 was inoperative, the global and/or server/dispatcher-specific binaries and configuration settings 900 may have changed within the central database 230. Accordingly, in one embodiment, when the node-specific bootstrap logic 905 is launched, it compares the binaries and configuration settings stored in the local file system of the server/dispatcher being started to the binaries and configuration settings 900 stored in the central database 230. In one embodiment, a comparison is performed between an index of the data stored locally on the server/dispatcher and an index of the data stored within the central database 320 (e.g., an index based on the hierarchical structure illustrated in **Figure 5**) to determine whether the configuration data at the server/dispatcher is out-of-date.

[0050] Regardless of how the comparison is performed, if the binaries and/or configuration settings stored within the local file system 904 of the dispatcher/ server are out-of-date, then the current binaries and/or configuration settings 900 are retrieved from the central database 230 and stored within the local file system 904 of the dispatcher/server. In one embodiment, only the binaries and/or configuration settings which are new are copied to the local file system 904, thereby conserving network bandwidth and server resources.

[0051] Once synchronization is complete, at 808, the startup and control logic 902 executes the processes on each of the servers using arguments/parameters included within the Instance Properties data object. At 810, the startup and

control logic 902 initializes the service framework and services on the servers/ dispatchers within the instance 210. For example, the service framework and services are the J2EE service framework and J2EE services, respectively, in a Java environment. However, various other types of services/frameworks may be employed while still complying with the underlying principles of the invention. Once the framework and services are running, the sever nodes are available to process service requests.

[0052] Embodiments of the invention may include various steps as set forth above. The steps may be embodied in machine-executable instructions which cause a general-purpose or special-purpose processor to perform certain steps. Alternatively, these steps may be performed by specific hardware components that contain hardwired logic for performing the steps, or by any combination of programmed computer components and custom hardware components.

[0053] Elements of the present invention may also be provided as a machine-readable medium for storing the machine-executable instructions. The machine-readable medium may include, but is not limited to, flash memory, optical disks, CD-ROMs, DVD ROMs, RAMs, EPROMs, EEPROMs, magnetic or optical cards, propagation media or other type of machine-readable media suitable for storing electronic instructions. For example, the present invention may be downloaded as a computer program which may be transferred from a remote computer (e.g., a server) to a requesting computer (e.g., a client) by way of data signals

embodied in a carrier wave or other propagation medium via a communication link (e.g., a modem or network connection).

[0054] Throughout the foregoing description, for the purposes of explanation, numerous specific details were set forth in order to provide a thorough understanding of the invention. It will be apparent, however, to one skilled in the art that the invention may be practiced without some of these specific details. For example, although the embodiments of the invention described above use a specific hierarchy within the configuration data object 500, various alternate organizational hierarchies may be defined while still complying with the underlying principles of the invention. For example, in one embodiment, a separate "binaries" configuration object is provided directly beneath the "cluster data" object (i.e., rather than rather than merely under each of the sub-configuration objects illustrated in **Figure 5**). Similarly, as an alternative to a "property sheet" for storing configuration parameters, files and name value pairs may also be used. For example, files may map a key name to a large group of data, whereas a name value pair may map a key name to a small group of data.

[0055] Accordingly, the scope and spirit of the invention should be judged in terms of the claims which follow.